NC STATE UNIVERSITY

Hytrace: A Hybrid Approach to Performance Bug Diagnosis in Production Cloud Infrastructures

Ting Dai, Daniel Dean†, Peipei Wang, Xiaohui Gu, Shan Lu*

†InsightFinder Inc. *University of Chicago

Motivation & Problem

- Performance bugs widely exist across different production server applications [25].
- Performance bugs are notoriously difficult to diagnose
- No runtime feedback (e.g., stack trace, error msg)
- No environment data
- Offline bug reproduction & debugging is **HARD**
- Previous work on performance bugs can be broadly classified into:

	Advantage	Disadvantage	Tradeoff
Static analysis [6, 7, 25, 36]	no runtime overhead	w/o focusing on the specific anomaly occurred, false positives	[25] specific types of performance problems, false negative
Dynamic runtime analysis [14, 20]	specific runtime problem more accurate	monitoring on production systems, runtime overhead.	[14, 20] base on system-level metrics without program semantics, cannot achieve precision.

Challenge

Static analysis

Infeasible to check:
1. Conf. happen long before call
2. Inter-procedural path-sensitive cannot scale production server

Too specific:
1. Low diagnosis coverage

Traditional loop detector not work:
1. Cannot explain why.

Dynamic runtime analysis

False positives:
1. Uncertainty of the statistical behavior modeling

False negatives:
1. The buggy function has abnormal manifestation at the system-level metrics or events.

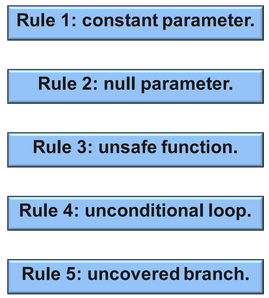
Application-level granularity from abnormal low-level system metrics or events:
1. Cannot achieve.

Components

Hytrace

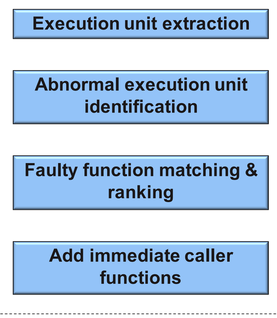
- Combine both **Hytrace-static** & **Hytrace-dynamic** to achieve high precision & coverage.
- Two carefully designed components are complementary to each other.
- Program semantic & run-time behavior info.

Hytrace-static



- Neither too specific nor too generic
- Differ from stand-alone static checkers
- **Generality over precision**, maybe-indicator not guarantee
- Similar like stand-alone static checkers
- Statically checkable without runtime info

Hytrace-dynamic



- Goal: **maximize coverage**
- Action: adding the immediate caller functions
- Intuition: caller function has abnormal practice, manifested in callee function.
- Identify root-cause functions but do not produce many system metrics.

Results Summary

The coverage & precision of different schemes

Bug Name	Hytrace TP	Hytrace FP	Infer(all) TP	Infer(all) FP	Infer(perf) TP	Infer(perf) FP	Findbugs(all) TP	Findbugs(all) FP	Findbugs(perf) TP	Findbugs(perf) FP	Caramel TP	Caramel FP	PerfScope TP	PerfScope FP
Apache-37680	✓	2	✗	18	✗	3	–	–	–	–	✗	4	✓	14
Apache-43238	✓	4	✗	18	✗	2	–	–	–	–	✗	2	✓	8
Apache-45856	✓	21	✗	18	✗	2	–	–	–	–	✗	4	✓	40
Lighttpd-1212	✓	0	✗	181	✗	61	–	–	–	–	✗	0	✓	0
Lighttpd-1999	✓	0	✗	171	✗	56	–	–	–	–	✗	0	✓	2
Memcached-106	✓	1	–	–	–	–	–	–	–	–	✗	0	✓	3
MySQL-54332	✓	0	–	–	–	–	–	–	–	–	✗	22	✓	2
MySQL-65615	✓	1	–	–	–	–	–	–	–	–	✗	8	✓	4
Cassandra-5064	✓	0	✓	2904	✓	2904	✗	322	✗	24	–	–	✓	3
Mapreduce-3738	✗	13	✓	5077	✓	5077	✗	1261	✗	170	–	–	✗	452
HDFS-3318	✓	26	✗	2367	✗	2367	✗	1401	✗	168	–	–	✓	276
Tomcat-53450	✓	0	✗	4638	✗	4638	✗	477	✗	53	–	–	✓	1
Tomcat-53173	✓	0	✗	4624	✗	4624	✗	422	✗	51	–	–	✓	12
Tomcat-42753	✓	11	–	–	–✗	–		889	✗	238	–	–	✓	29

Coverage comparison for all performance bugs

System Name	Total # bugs	Coverage: Hytrace	Infer(all)	Infer(perf)	Findbugs(all)	Findbugs(perf)	Caramel	Num. of bugs matched by each rule: R1	R2	R3	R4	R5
Apache	13	100%	0%	0%	–	–	0%	11	10	3	3	13
Lighttpd	7	100%	0%	0%	–	–	0%	6	2	0	2	7
Memcached	1	100%	0%	0%	–	–	0%	0	1	0	0	1
MySQL	19	100%	11%	5%	–	–	5%	18	18	2	7	17
Squid	13	100%	0%	0%	–	–	0%	13	6	0	1	13
Cassandra	27	100%	44%	44%	0%	37%	–	9	3	–	26	1
HDFS	19	100%	39%	39%	0%	17%	–	13	5	–	17	7
Mapreduce	27	100%	59%	59%	48%	57%	–	20	12	–	24	14
Tomcat	7	100%	43%	43%	14%	43%	–	6	2	–	4	2

Coverage comparison for all performance bugs

Bug Name	Hytrace Rank	R1	R2	R3	R4	R5	PerfScope Rank
Apache-37680	3	✓	✓	✗	✗	✓	15
Apache-43238	5	✓	✓	✗	✗	✓	9
Apache-45856	22	✓	✗	✓	✗	✓	41
Lighttpd-1212	1	✓	✗	✗	✗	✗	1
Lighttpd-1999	1	✓	✗	✗	✗	✓	3
Memcached-106	2	✓	✓	✗	✗	✓	4
MySQL-54332	1	✓	✓	✗	✗	✗	3
MySQL-65615	2	✓	✗	✗	✗	✓	5
Cassandra-5064	1	✓	✗	✗	✓	✗	4
Mapreduce-3738	✗	✓	✓	✗	✗	✗	✗
HDFS-3318	27	✓	✓	✗	✗	✓	277
Tomcat-53450	1	✗	✗	✗	✓	✗	2
Tomcat-53173	1	✗	✗	✗	✗	✓	13
Tomcat-42753	12	✓	✓	✗	✗	✗	30

Overhead of HyTrace-static & HyTrace-dynamic

Bug Name	Static analysis time (sec)	App LOC(K)	Dynamic analysis time(min)	Trace size(MB)
Apache-37680	5.9±0.02	266.7	1.3±0.01	406
Apache-43238	4.5±0.01	312.8	3.5±0.01	306
Apache-45856	4.8±0.02	314.7	2.4±0.01	324
Lighttpd-1212	2.1±0.01	53.9	1.1±0.01	337
Lighttpd-1999	3.0±0.01	58.4	13.1±0.02	1,365
Memcached-106	29.2±0.01	11.0	25.6±0.32	3,603
MySQL-54332	9.6±0.01	1,233	9.2±0.41	316
MySQL-65615	12.9±0.03	1,759	5.8±0.12	77
Cassandra-5064	29.2±2.23	259.0	21.7±0.40	1,054
Mapreduce-3738	40.5±3.02	935.8	16.0±0.20	550
HDFS-3318	108±0.60	1,114	15.7±1.22	473
Tomcat-53450	35.2±0.38	407.9	5.7±0.34	35
Tomcat-53173	49.7±0.40	405.0	2.0±0.02	143
Tomcat-42753	49.5±0.20	456.9	9.1±0.80	274
Avg.	27.4±0.50	542.0	9.4±0.28	662

Conclusion

- Hytrace combines offline static analysis and online dynamic bug inference.
- Hytrace does not require any application source code or instrumentation.
- We have implemented a prototype of Hytrace and tested it over a set of real performance bugs.
- Our results show that Hytrace can significantly reduce false positive functions and significantly improve the rank of bug related functions.
- Even though Hytrace-static is still preliminary (with only 5 rules), it can cover all the 133 performance bugs we sampled from bug reports.
- Hytrace failed to identify root cause related functions for 1 out of 14 bugs we studied due to the false negative of Hytrace-dynamic component.
- We believe the false negatives can be addressed by using advanced Hytrace dynamic mechanism.
- Hytrace imposes less than 3% CPU overhead to production cloud environments.

Supported in part by NSF CNS1513942 grant and NSF CNS1149445 grant.